

PCI9018 数据采集卡

WIN2000/XP 驱动程序使用说明书



阿尔泰科技发展有限公司

产品研发部修订

请您务必阅读《[使用纲要](#)》，他会使您事半功倍!

目 录

目 录.....	1
第一章 版权信息与命名约定.....	2
第一节、版权信息.....	2
第二节、命名约定.....	2
第二章 使用纲要.....	2
第一节、使用上层用户函数，高效、简单.....	2
第二节、如何管理设备.....	2
第三节、如何用 Dma 直接内存方式取得 AD 数据.....	3
第四节、哪些函数对您不是必须的.....	3
第三章 PCI 设备操作函数接口介绍.....	5
第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI9018_”）.....	6
第二节、设备对象管理函数原型说明.....	7
第三节、AD 采样操作函数原型说明.....	10
第四节、AD 硬件参数保存与读取函数原型说明.....	14
第四章 硬件参数结构.....	15
第一节、AD 硬件参数介绍（PCI9018_PARA_AD）.....	15
第二节、AD 状态参数结构（PCI9018_STATUS_AD）.....	17
第五章 数据格式转换与排列规则.....	18
第一节、AD 原码 LSB 数据转换成电压值的换算方法.....	18
第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则.....	19
第三节、AD 测试应用程序创建并形成的数据文件格式.....	19
第六章 上层用户函数接口应用实例.....	20
第一节、简易程序演示说明.....	20
第二节、高级程序演示说明.....	20
第七章 高速大容量、连续不间断数据采集及存盘技术详解.....	20
第八章 共用函数介绍.....	22
第一节、公用接口函数总列表（每个函数省略了前缀“PCI9018_”）.....	22
第二节、PCI 内存映射寄存器操作函数原型说明.....	22
第三节、IO 端口读写函数原型说明.....	30
第四节、线程操作函数原型说明.....	33
第五节、文件对象操作函数原型说明.....	34

第一章 版权信息与命名约定

第一节、版权信息

本软件产品及相关套件均属北京阿尔泰科技发展有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位、我公司授权的代理商及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。若您需要我公司产品及相关信息请及时与当地代理商联系或直接与我们联系，我们将热情接待。

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCIxxxx_则被省略。如 PCI9018_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写规则

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注: 在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

以上规则不局限于该产品。

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如 Win32 API 的函数，具有相当的灵活性、可靠性和高效性。诸如 [InitDeviceAD](#)、[WriteRegisterULong](#)、[ReadRegisterULong](#)、[WritePortByte](#)、[ReadPortByte](#)……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上插座等管脚分配情况。

第二节、如何管理设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 [CreateDevice](#) 函数创建一个设备对象句柄 `hDevice`，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，如 [InitDeviceAD](#) 可以使用 `hDevice` 句柄以程序查询方式初始化设备的 AD 部件，[ReadDeviceAD](#) 函数可以用 `hDevice` 句柄实现对 AD 数据的采样读取等。最后可以通过 [ReleaseDevice](#) 将 `hDevice` 释放掉。

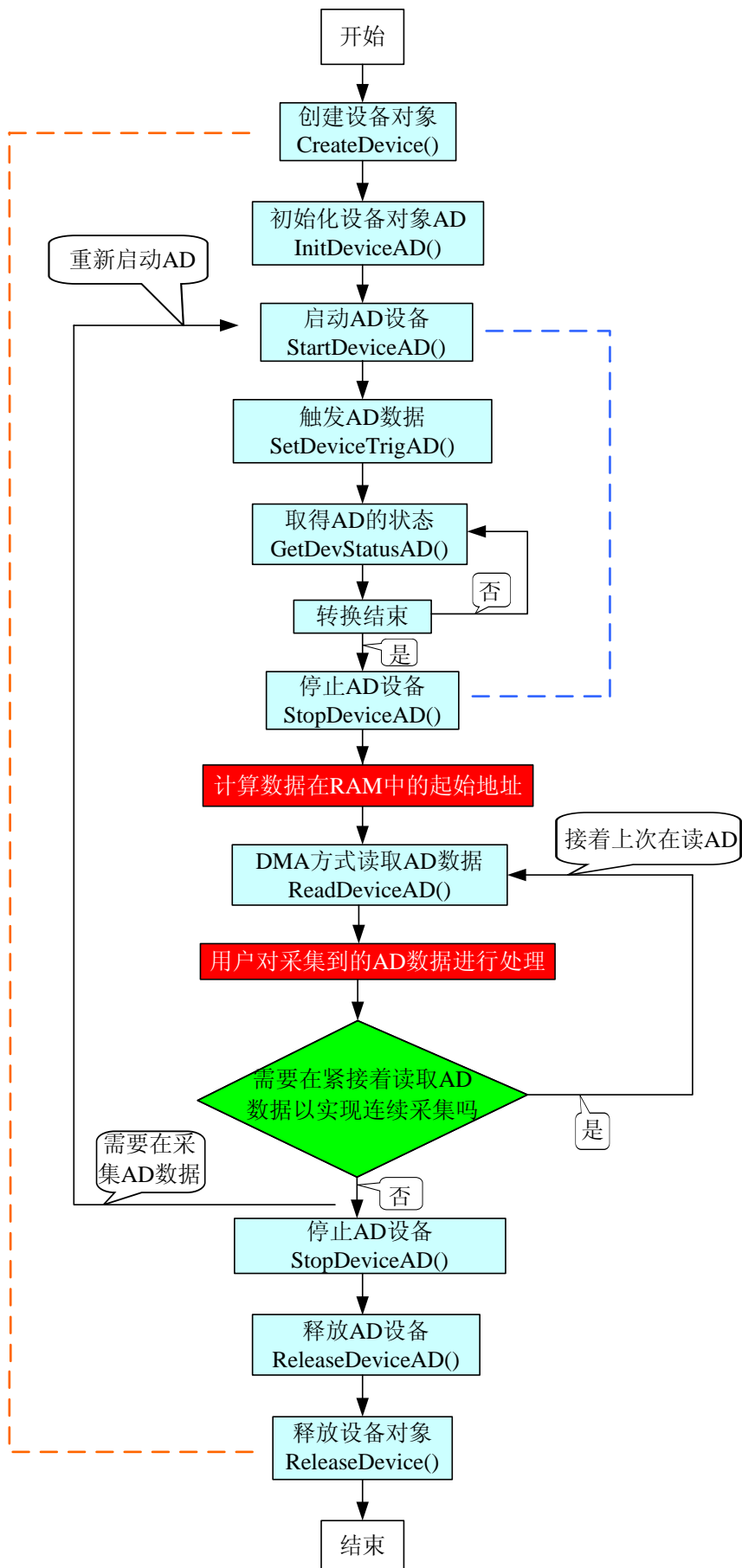
第三节、如何用 Dma 直接内存方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用[InitDeviceAD](#)函数初始化AD部件，关于采样通道、频率等的参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用[StartDeviceAD](#)即可启动AD部件，开始AD采样，当您需要暂停设备时，执行[StopDeviceAD](#)，当您需要关闭AD设备时，[ReadDeviceAD](#)便可帮您实现（但设备对象hDevice依然存在）。具体执行流程请看图 2.1.1。

注意：图中较粗的虚线表示对称关系。如红色虚线表示[CreateDevice](#)和[ReleaseDevice](#)两个函数的关系是：最初执行一次[CreateDevice](#)，在结束是就须执行一次[ReleaseDevice](#)。

第四节、哪些函数对您不是必须的

公共函数如[CreateFileObject](#)，[WriteFile](#)，[ReadFile](#)等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么[WriteRegisterByte](#)，[WriteRegisterWord](#)，[WriteRegisterULong](#)，[ReadRegisterByte](#)，[ReadRegisterWord](#)，[ReadRegisterULong](#)等函数您可完全不必理会，除非您是作为底层用户管理设备。而[WritePortByte](#)，[WritePortWord](#)，[WritePortULong](#)，[ReadPortByte](#)，[ReadPortWord](#)，[ReadPortULong](#)则对PCI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。



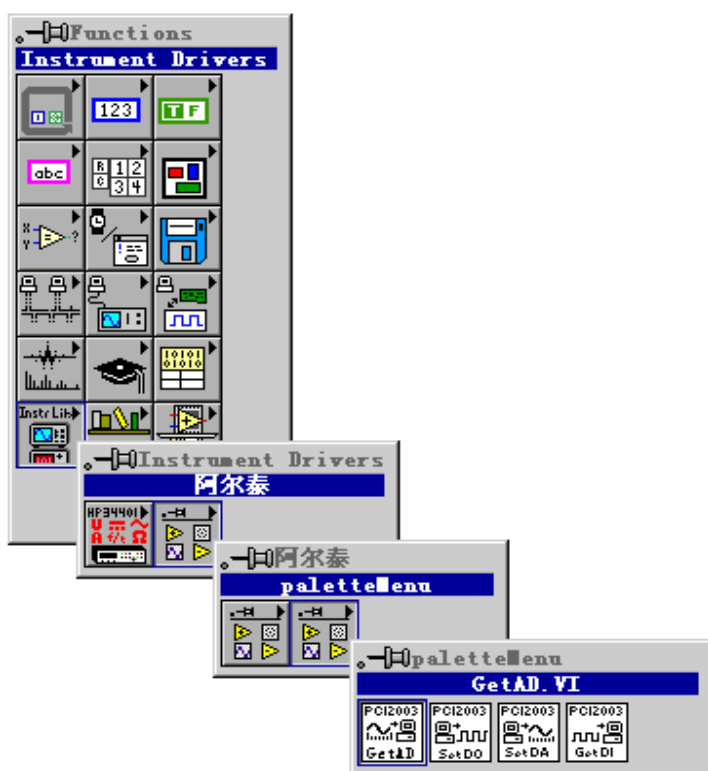
2.1.1 DMA 方式 AD 采集实现过程

第三章 PCI 设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如[InitDeviceProAD](#)）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用[ReadDeviceProAD Npt](#)（或[ReadDeviceProAD Half](#)）函数指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的Bit位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉PCI总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解PCI的资源配置空间、PNP即插即用管理，而只须用[GetDeviceAddr](#)函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用[ReadRegisterULong](#)和[WriteRegisterULong](#)对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《[设备驱动接口函数总列表](#)》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Library Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI9018_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PCI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层及底层用户
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备, 且释放 PCI 总线设备对象	上层及底层用户
② AD 读取函数		
ADCalibration	校准函数	上层用户
InitDeviceAD	初始化 AD 部件, 如通道等	上层用户
StartDeviceAD	启动 AD 采集	上层用户
SetDeviceTrigAD	产生软件触发事件	上层用户
GetDevStatusAD	取得 DMA 的各种状态	上层用户
ReadDeviceAD	DMA 方式读 AD 状态	
StopDeviceAD	停止 AD 采集	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
③ AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户

使用需知:**Visual C++ & C++Builder:**

要使用如下函数关键的问题是:

首先, 必须在您的源程序中包含如下语句:

```
#include "C:\Art\PCI9018\INCLUDE\PCI9018.H"
```

注: 以上语句采用默认路径和默认板号, 应根据您的板号和安装情况确定 PCI9018.H 文件的正确路径, 当然也可以把此文件拷到您的源程序目录中。

另外, 要在 VB 环境中用子线程以实现高速、连续数据采集与存盘, 请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版, 也可以实现子线程操作。

C++ Builder:

要使用如下函数一个关键的问题是首先必须将我们提供的头文件(PCI9018.H)写进您的源程序头部。如: #include “\Art\PCI9018\Include\PCI9018.h”, 然后再将 PCI9018.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”(Add to Project)命令, 在弹出的对话框中分别选择文件类型: Library file (*.lib), 即可选择 PCI9018.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下。

Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单, 执行其中的“添加模块”(Add Module)命令, 在弹出的对话框中选择 PCI9018.Bas 模块文件, 该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

请注意, 因考虑 Visual C++和 Visual Basic 两种语言的兼容问题, 在下列函数说明和示范程序中, 所举的 Visual Basic 程序均是需编译后在独立环境中运行。所以用户若在解释环境中运行这些代码, 我们不能保证完全顺利运行。

Delphi:

要使用如下函数一个关键的问题是首先必须将我们提供的单元模块文件 (*.Pas) 加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单, 执行其中的“Project Manager”命令, 在弹出的对话框中选择*.exe 项目, 再单击鼠标右键, 最后 Add 指令, 即可将 PCI9018.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中, 执行 Add To Project 命令, 然后选择*.Pas 文件类型也能实现单元模块文件的添加。该文件

的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PCI9018”。如：

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
PCI9018; // 注意： 在此加入驱动程序接口单元 PCI9018

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：



- 一、在 LabView 中打开 PCI9018.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。
- 二、根据 LabView 语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，如 ReadDeviceProAD 接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。
- 三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“[U16]”为无符号 16 位短整型数组或缓冲区或指针，“[U32]”与 “[U16]”同理，只是位数不一样。

第二节、设备对象管理函数原型说明

◆ 创建设备对象函数（逻辑号）

函数原型：

Visual C++ & C++Builder :

HANDLE CreateDevice (int DeviceID = 0)

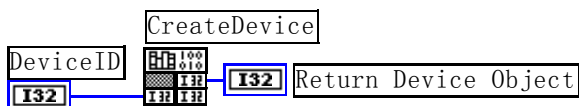
Visual Basic :

Declare Function CreateDevice Lib "PCI9018" (Optional ByVal DeviceID As Integer = 0) As Long

Delphi :

Function CreateDevice(DeviceID : Integer = 0) : Integer;
StdCall; External 'PCI9018' Name 'CreateDevice ';

LabVIEW :



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对设备所有功能的访问。

参数：

DeviceID 设备 ID(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，系统将以该设备的“基本名称”与 DeviceID 标识值为名称后缀的标识符来确认和管理该设备。默认值为 0。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： [CreateDevice](#) [GetDeviceCount](#) [GetDeviceCurrentID](#)
[ListDeviceDlg](#) [ReleaseDevice](#)

Visual C++ & C++Builder 程序举例

```

:
HANDLE hDevice; // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PCI9018_CreateDevice (DeviceLgcID); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALID_HANDLE_VALUE); // 判断设备对象句柄是否有效
{
    return; // 退出该函数
}

```

Visual Basic 程序举例

```

:
Dim hDevice As Long ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PCI9018_CreateDevice (DeviceLgcID) ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then ' 判断设备对象句柄是否有效
    MsgBox "创建设备对象失败"
    Exit Sub ' 退出该过程
End If
:

```

◆ 取得本计算机系统中 PCI9018 设备的总数量

函数原型:

Visual C++ & C++Builder:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

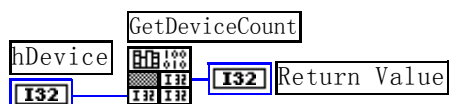
Declare Function GetDeviceCount Lib "PCI9018" (ByVal hDevice As Long) As Integer

Delphi:

```

Function GetDeviceCount (hDevice : Integer) : Integer;
    StdCall; External 'PCI9018' Name ' GetDeviceCount ';

```

LabVIEW:**功能:** 取得 PCI9018 设备的数量。**参数:**hDevice设备对象句柄, 它应由[CreateDevice](#)创建。**返回值:** 返回系统中 PCI9018 的数量。

相关函数: [CreateDevice](#) [GetDeviceCount](#) [GetDeviceCurrentID](#)
 [ListDeviceDlg](#) [ReleaseDevice](#)

◆ 取得该设备当前逻辑 ID 和物理 ID

函数原型:

Visual C++ & C++Builder:

```

BOOL GetDeviceCurrentID (HANDLE hDevice,
                        PLONG DeviceLgcID,
                        PLONG DevicePhysID)

```

Visual Basic:

```

Declare Function GetDeviceCurrentID Lib "PCI9018" (ByVal hDevice As Long, _
                                                ByRef DeviceLgcID As Long, _

```

Delphi:

```
Function GetDeviceCurrentID (hDevice : Integer;
                             DeviceLgcID : Pointer;
                             DevicePhysID : Pointer) : Boolean;
StdCall; External 'PCI9018' Name 'GetDeviceCurrentID';
```

LabVIEW:

请参考相关演示程序。

功能: 取得指定设备逻辑和物理 ID 号。

参数:

hDevice 设备对象句柄，它指向要取得逻辑和物理号的设备，它应由[CreateDevice](#)创建。

DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。

DevicePhysID 返回设备的物理 ID，它的取值范围为[0, 15]，它的具体值由卡上的拨码器 DID 决定。

返回值: 如果初始化设备对象成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数: [CreateDevice](#) [GetDeviceCount](#) [GetDeviceCurrentID](#)
 [ListDeviceDlg](#) [ReleaseDevice](#)

◆ 用对话框控件列表计算机系统中所有 PCI9018 设备各种配置信息

函数原型:

Visual C++ & C++Builder:

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function ListDeviceDlg Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

```
Function ListDeviceDlg (hDevice : Integer) : Boolean;
StdCall; External 'PCI9018' Name 'ListDeviceDlg';
```

LabVIEW:

请参考相关演示程序。

功能: 列表系统中 PCI9018 的硬件配置信息。

参数:

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

返回值: 若成功，则弹出对话框控件列表所有 PCI9018 设备的配置情况。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 释放设备对象所占的系统资源及设备对象

函数原型:

Visual C++ & C++Builder:

BOOL ReleaseDevice(HANDLE hDevice)

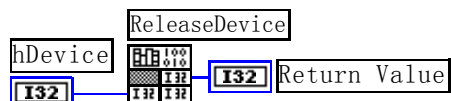
Visual Basic:

Declare Function ReleaseDevice Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

```
Function ReleaseDevice(hDevice : Integer) : Boolean;
StdCall; External 'PCI9018' Name 'ReleaseDevice';
```

LabVIEW:



功能: 释放设备对象所占用的系统资源及设备对象自身。

参数: `hDevice`设备对象句柄, 它应由[CreateDevice](#)创建。

返回值: 若成功, 则返回TRUE, 否则返回FALSE, 用户可以用[GetLastErrorEx](#)捕获错误码。

相关函数: [CreateDevice](#)

应注意的是, [CreateDevice](#)必须和[ReleaseDevice](#)函数一一对应, 即当您执行了一次[CreateDevice](#)后, 再一次执行这些函数前, 必须执行一次[ReleaseDevice](#)函数, 以释放由[CreateDevice](#)占用的系统软硬件资源, 如DMA控制器、系统内存等。只有这样, 当您再次调用[CreateDevice](#)函数时, 那些软硬件资源才可被再次使用。

第三节、AD 采样操作函数原型说明

◆ 设备校准函数

函数原型:

Visual C++ & C++Builder:

BOOL ADCalibration(HANDLE hDevice)

Visual Basic:

Declare Function ADCalibration Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

Function ADCalibration (hDevice : Integer) : Boolean;

StdCall; External 'PCI9018' Name ' ADCalibration ';

LabVIEW:

请参考相关演示程序。

功能: 设备校准函数。

参数: `hDevice`设备对象句柄, 它应由[CreateDevice](#)创建。

返回值: 若成功, 则返回TRUE, 否则返回FALSE, 用户可以用[GetLastErrorEx](#)捕获错误码。

相关函数: [CreateDevice](#)

◆ 初始化设备对象

函数原型:

Visual C++ & C++Builder:

BOOL InitDeviceAD(HANDLE hDevice,
PCI9018_PARA_AD pADPara)

Visual Basic:

Declare Function InitDeviceAD Lib "PCI9018" (ByVal hDevice As Long, _
ByRef pADPara As PCI9018_PARA_AD) As Boolean

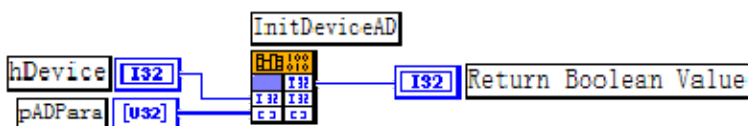
Delphi:

Function InitDeviceAD(hDevice : Integer;

pADPara:PCI9018_PARA_AD):Boolean;

StdCall; External 'PCI9018' Name ' InitDeviceAD ';

LabVIEW:



功能: 它负责初始化设备对象中的AD部件, 为设备操作就绪有关工作, 然后启动AD设备开始AD采集, 随后, 用户便可以连续调用[ReadDeviceAD](#)读取设备上的AD数据以实现连续采集。

参数:

hDevice 设备对象句柄, 它应由设备的[CreateDevice](#)创建。

pADPara 设备对象参数结构, 它决定了设备对象的各种状态及工作方式。请参考《[AD硬件参数介绍](#)》。

返回值: 如果初始化设备对象成功, 则返回TRUE, 且AD便被启动。否则返回FALSE, 用户可用[GetLastErrorEx](#)捕获当前错误码, 并加以分析。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

注意: 该函数将试图占用系统的某些资源, 如系统内存区、DMA资源等。所以当用户在反复进行数据采集之前, 只须执行一次该函数即可, 否则某些资源将会发生使用上的冲突, 便会失败。除非用户执行了[ReleaseDeviceAD](#)函数后, 再重新开始设备对象操作时, 可以再执行该函数。所以该函数切忌不要单独放在循环语句中反复执行, 除非和[ReleaseDeviceAD](#)配对。

◆ 启动 AD 设备(Start device AD for program mode)

函数原型:

Visual C++ & C++Builder:

BOOL StartDeviceAD (HANDLE hDevice)

Visual Basic:

Declare Function StartDeviceAD Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

Function StartDeviceAD (hDevice : Integer): Boolean;

StdCall; External 'PCI9018' Name ' StartDeviceAD ';

LabVIEW:

请参考相关演示程序。

功能: 启动AD设备, 它必须在调用[InitDeviceAD](#)后才能调用此函数。该函数除了启动AD设备开始转换以外, 不改变设备的其他任何状态。

参数:

hDevice 设备对象句柄, 它应由[CreateDevice](#)创建。

返回值: 如果调用成功, 则返回TRUE, 且AD立刻开始转换, 否则返回FALSE, 用户可用[GetLastErrorEx](#)捕获当前错误码, 并加以分析。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 产生软件触发事件

函数原型:

Visual C++ & C++Builder:

BOOL SetDeviceTrigAD (HANDLE hDevice)

Visual Basic:

Declare Function SetDeviceTrigAD Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

Function SetDeviceTrigAD (hDevice : Integer): Boolean;

StdCall; External 'PCI9018' Name ' SetDeviceTrigAD ';

LabVIEW:

请参考相关演示程序。

功能: 当设备使能允许后, 产生软件触发事件 (只有触发源为软件触发时有效)。

参数: **hDevice** 设备对象句柄, 它应由[CreateDevice](#)创建。

返回值: 如果调用成功, 则返回TRUE, 否则返回FALSE, 用户可用[GetLastErrorEx](#)捕获当前错误码, 并加以分析。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 取得 AD 状态标志

函数原型:

Visual C++ & C++Builder:

```
BOOL GetDevStatusAD (HANDLE hDevice,
                    PCI9018_STATUS_AD pADStatus);
```

Visual Basic:

```
Declare Function GetDevStatusAD Lib "PCI9018" (ByVal hDevice As Long, _
                                             ByRef pADStatus As PCI9018_STATUS_AD) As Boolean
```

Delphi:

```
Function GetDevStatusAD (hDevice : Integer;
                        pADStatus : PCI9018_STATUS_AD) : Boolean;
StdCall; External 'PCI9018' Name ' GetDevStatusAD ';
```

LabVIEW:

请参考相关演示程序。

功能: 一旦用户使用[StartDeviceAD](#)后, 应立即用此函数查询存储器的状态。

参数:

hDevice 设备对象句柄, 它应由[CreateDevice](#)创建。

pADStatus 获得AD的各种当前状态。它属于结构体, 具体定义请参考《[AD状态参数结构 \(PCI9018_STATUS_AD\)](#)》章节。

返回值: 若调用成功则返回TRUE, 否则返回FALSE, 用户可以调用[GetLastErrorEx](#)函数取得当前错误码。

相关函数:

CreateDevice	InitDeviceAD	StartDeviceAD
SetDeviceTrigAD	GetDevStatusAD	ReadDeviceProAD
StopDeviceAD	ReleaseDeviceAD	ReleaseDevice

◆ DMA 方式读取设备上的 AD 数据

函数原型:

Visual C++ & C++Builder:

```
BOOL ReadDeviceAD (HANDLE hDevice,
                  PWORD pADBuffer,
                  ULONG nReadSizeWords,
                  PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD Lib "PCI9018" (ByVal hDevice As Long, _
                                             ByRef pADBuffer As Integer, _
                                             ByVal nReadSizeWords As Long, _
                                             ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD (hDevice : Integer;
                        pADBuffer : Pointer;
                        nReadSizeWords : LongInt;
                        nRetSizeWords : Pointer) : Boolean;
StdCall; External 'PCI9018' Name ' ReadDeviceProAD ';
```

LabVIEW:

请参考相关演示程序。

功能: 程序方式读 AD 数据。

参数:

hDevice 设备对象句柄, 它应由[CreateDevice](#)创建。

pADBuffer将用于接受数据的用户缓冲区(该区必须开辟大于M加N个字的空间)。关于如何将这AD数据转换成相应的电压值, 请参考《[数据格式转换与排列规则](#)》。

nReadSizeWords 指定一次 ReadDeviceProAD 操作应读取多少字数据到用户缓冲区。必须等于 M 加 N 的长度。

nRetSizeWords 返回实际读取的数据长度。

返回值: 其返回值表示所成功读取的数据点数(字),也表示当前读操作在ADBuffer缓冲区中的有效数据量。通常情况下其返回值应与ReadSizeWords参数指定量的数据长度(字)相等,除非用户在这个读操作以外的其他线程中执行了ReleaseDeviceAD函数中断了读操作,否则设备可能有问题。对于返回值不等于nReadSizeWords参数值的,用户可用GetLastErrorEx捕获当前错误码,并加以分析。

注释:此函数也可用于单点读取和几个点的读取,只需要将nReadSizeWords设置成1或相应值即可。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 暂停 AD 设备

函数原型:

Visual C++ & C++Builder:

BOOL StopDeviceAD (HANDLE hDevice)

Visual Basic:

Declare Function StopDeviceAD Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

Function StopDeviceAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI9018' Name ' StopDeviceAD ';

LabVIEW:

请参考相关演示程序。

功能: 暂停AD设备。它必须在调用StartDeviceAD后才能调用此函数。该函数除了停止AD设备不再转换以外,不改变设备的其他任何状态。此后您可再调用StartDeviceAD函数重新启动AD,此时AD会按照暂停以前的状态(如FIFO存储器位置、通道位置)开始转换。

参数:

hDevice 设备对象句柄,它应由CreateDevice创建。

返回值: 如果调用成功,则返回TRUE,且AD立刻停止转换,否则返回FALSE,用户可用GetLastErrorEx捕获当前错误码,并加以分析。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 释放设备上的 AD 部件

函数原型:

Visual C++ & C++ Builder:

BOOL ReleaseDeviceAD(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceAD Lib "PCI9018" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDeviceAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI9018' Name ' ReleaseDeviceAD ';

LabVIEW:

请参考相关演示程序。

功能: 释放设备上的AD部件。

参数: **hDevice** 设备对象句柄,它应由CreateDevice创建。

返回值: 若成功,则返回TRUE,否则返回FALSE,用户可以用GetLastErrorEx捕获错误码。

应注意的是,InitDeviceAD必须和ReleaseDeviceAD函数一一对应,即当您执行了一次InitDeviceAD后,再一次执行这些函数前,必须执行一次ReleaseDeviceAD函数,以释放由InitDeviceAD占用的系统软硬件资源,如映射寄存器地址、系统内存等。只有这样,当您再次调用InitDeviceAD函数时,那些软硬件资源才可被再次使用。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [ReleaseDeviceAD](#)

ReleaseDevice

◆ 函数一般调用顺序

- ① [CreateDevice](#)
- ② [InitDeviceAD](#)
- ③ [StartDeviceAD](#)
- ④ [GetDevStatusAD](#)
- ⑤ [ReadDeviceAD](#)
- ⑥ [StopDeviceAD](#)
- ⑦ [ReleaseDeviceAD](#)
- ⑧ [ReleaseDevice](#)

注明: 用户可以反复执行第⑤步, 以实现高速连续不间断大容量采集。

关于这个过程的图形说明请参考《[使用纲要](#)》。

注意: 若成功初始化 DMA 后, 要退出整个应用程序, 切记应先释放 DMA 才能退出。

第四节、AD 硬件参数保存与读取函数原型说明

◆ 从 Windows 系统中读入硬件参数函数

函数原型:

Visual C++ & C++ Builder:

```
BOOL LoadParaAD(HANDLE hDevice,
                PCI9018_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "PCI9018" (ByVal hDevice As Long, _
                                           ByRef pADPara As PCI9018_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD (hDevice : Integer;
                    pADPara : PCI9018_PARA_AD) : Boolean;
StdCall; External 'PCI9018' Name 'LoadParaAD';
```

LabVIEW:

请参考相关演示程序。

功能: 负责从 Windows 系统中读取设备的硬件参数。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

pADPara 属于 PCI9018_PARA_AD 的结构指针类型, 它负责返回 PCI 硬件参数值, 关于结构指针类型 PCI9018_PARA_AD 请参考 PCI9018.h 或 PCI9018.Bas 或 PCI9018.Pas 函数原型定义文件, 也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [LoadParaAD](#) [SaveParaAD](#)
[ReleaseDevice](#)

◆ 往 Windows 系统写入设备硬件参数函数

函数原型:

Visual C++ & C++ Builder:

```
BOOL SaveParaAD (HANDLE hDevice,
                PCI9018_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PCI9018" (ByVal hDevice As Long, _
                                           ByRef pADPara As PCI9018_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;
```

```
pADPara : PCI9018_PARA_AD) : Boolean;  
StdCall; External 'PCI9018' Name ' SaveParaAD ';
```

LabVIEW:

请参考相关演示程序。

功能: 负责把用户设置的硬件参数保存在 Windows 系统中, 以供下次使用。

参数:

hDevice设备对象句柄, 它应由[CreateDevice](#)创建。

pADPara设备硬件参数, 关于PCI9018_PARA_AD的详细介绍请参考PCI9018.h或PCI9018.Bas或PCI9018.Pas函数原型定义文件, 也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [LoadParaAD](#) [SaveParaAD](#)
 [ReleaseDevice](#)

◆ **AD 采样参数复位至出厂默认值函数**

函数原型:

Visual C++ & C++ Builder:

```
BOOL ResetParaAD (HANDLE hDevice,  
                  PCI9018_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "PCI9018" (ByVal hDevice As Long, _  
                                          ByRef pADPara As PCI9018_PARA_AD) As Boolean
```

Delphi:

```
Function ResetParaAD (hDevice : Integer;  
                      pADPara : PCI9018_PARA_AD) : Boolean;  
StdCall; External 'PCI9018' Name ' ResetParaAD ';
```

LabVIEW:

请参考相关演示程序。

功能: 将系统中原来的 AD 参数值复位至出厂时的默认值。以防用户不小心将各参数设置错误造成一时无法确定错误后果的后果。

参数:

hDevice设备对象句柄, 它应由[CreateDevice](#)创建。

pADPara设备硬件参数, 它负责在参数被复位后返回其复位后的值。关于PCI9018_PARA_AD的详细介绍请参考PCI9018.h或PCI9018.Bas或PCI9018.Pas函数原型定义文件, 也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [LoadParaAD](#) [SaveParaAD](#)
 [ResetParaAD](#) [ReleaseDevice](#)

注意: 在您编写工程应用软件时, 若要更方便的保存和读取您特有的软件参数, 请不防使用我们为您提供的辅助函数: [SaveParaInt](#)、[LoadParaInt](#)、[SaveParaString](#)、[LoadParaString](#), 详细说明请参考共用函数介绍章节中的《[各种参数保存和读取函数原型说明](#)》。

第四章 硬件参数结构

第一节、AD 硬件参数介绍 (PCI9018_PARA_AD)

Visual C++ & C++Builder:

```
typedef struct _PCI9018_PARA_AD        // 板卡各参数值  
{  
    LONG InputRange[2];                // 模拟量输入量程选择  
    LONG Frequency;                    // 采集频率,单位为 Hz, [1, 80000]
```



```

LONG Gains[4];           // 增益控制
LONG M_Length;          // M 段长度(字), 总的取值范围 1-32M, 但是 M 加 N 长度不能大于 32M
LONG N_Length;          // N 段长度(字), 总的取值范围 1-32M, 但是 M 加 N 长度不能大于 32M
LONG TriggerMode;       // 触发模式选择
LONG TriggerDir;        // 触发方向选择(正向/负向触发)
LONG bAvailableTrig;    // 当触发事件提前时, 是否有效? (=TRUE:有效, =FALSE:忽略)
LONG TriggerSource;     // 触发源选择
LONG TrigLevelVolt;     // 触发电平(0-10000mV)
LONG ClockSource;       // 时钟源选择(内/外时钟源)
LONG bClockOutput;     // 允许时钟输出到 CLKOUT,=TRUE:允许时钟输出, =FALSE:禁止时钟输出
} PCI9018_PARA_AD, *PCI9018_PARA_AD;

```

Visual Basic:

```

Private Type PCI9018_PARA_AD
    InputRange (0 to 1) As Long ' 模拟量输入量程选择
    Frequency As Long           ' 采集频率,单位为 Hz, [1, 80000]
    Gains[4] As Long           ' 增益控制
    M_Length As Long           ' M 段长度(字)
    N_Length As Long           ' N 段长度(字)
    TriggerMode As Long        ' 触发模式选择
    TriggerDir As Long         ' 触发方向选择(正向/负向触发)
    bAvailableTrig As Long     ' 当触发事件提前时, 是否有效? (=TRUE:有效, =FALSE:忽略)
    TriggerSource As Long      ' 触发源选择
    TrigLevelVolt As Long      ' 触发电平(0-10000mV)
    ClockSource As Long        ' 时钟源选择(内/外时钟源)
    bClockOutput As Long      ' 允许时钟输出到 CLKOUT,=TRUE:允许时钟输出, =FALSE:禁止时钟输出
End Type

```

Delphi:

```

Type // 定义结构体数据类型
PCI9018_PARA_AD = ^PCI9018_PARA_AD; // 指针类型结构
PCI9018_PARA_AD = record // 标记为记录型
    InputRange : Array[0...1] of LongInt; // 模拟量输入量程选择
    Frequency : LongInt; // 采集频率,单位为 Hz, [1, 80000]
    Gains[4] : LongInt; // 增益控制
    M_Length: LongInt; // M 段长度(字)
    N_Length: LongInt; // N 段长度(字)
    TriggerMode : LongInt; // 触发模式选择
    TriggerDir: LongInt; // 触发方向选择(正向/负向触发)
    bAvailableTrig: LongInt; // 当触发事件提前时, 是否有效? (=TRUE:有效, =FALSE:忽略)
    TriggerSource : LongInt; // 触发源选择
    TrigLevelVolt : LongInt; // 触发电平(0-10000mV)
    ClockSource : LongInt; // 时钟源选择(内/外时钟源)
    bClockOutput: LongInt; // 允许时钟输出到 CLKOUT,=TRUE:允许时钟输出, =FALSE:禁止时钟输出
End;

```

LabVIEW:

请参考相关演示程序。

此结构主要用于设定设备AD硬件参数值, 用这个参数结构对设备进行硬件配置完全由[InitDeviceAD](#)函数自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

bChannelArray 采样通道选择阵列, 分别控制 16 个通道, =TRUE 表示该通道采样, 否则不采样。

InputRange 模拟量输入量程选择。

常量名	常量值	功能定义
PCI9018_INPUT_N1000_P10000mV	0x00	±10000mV

Frequency 采集频率，单位为 Hz，取值范围为[1, 80000]。

M_Length M 段长度(字)，总的取值范围 1-32M，但是 M 加 N 长度不能大于 32M。

N_Length N 段长度(字)，总的取值范围 1-32M，但是 M 加 N 长度不能大于 32M。

Gain AD 采样程控增益。

常量名	常量值	功能定义
PCI9018_GAINS_1MULT	0x00	1 倍增益
PCI9018_GAINS_2MULT	0x01	2 倍增益
PCI9018_GAINS_4MULT	0x02	5 倍增益
PCI9018_GAINS_8MULT	0x03	10 倍增益

TriggerMode AD 触发模式。

常量名	常量值	功能定义
PCI9018_TRIGMODE_MIDL	0x00	中间触发
PCI9018_TRIGMODE_POST	0x01	后触发
PCI9018_TRIGMODE_PRE	0x02	预触发
PCI9018_TRIGMODE_DELAY	0x03	硬件延时触发

TriggerSource AD 触发源。

常量名	常量值	功能定义
PCI9018_TRIGMODE_SOFT	0x00	软件触发
PCI9018_TRIGSRC_DTR	0x01	选择 DTR 作为触发源
PCI9018_TRIGSRC_ATR	0x02	选择 ATR 作为触发源

TriggerDir AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义
PCI9018_TRIGDIR_NEGATIVE	0x00	负向触发（下降沿触发）
PCI9018_TRIGDIR_POSITIVE	0x01	正向触发（上升沿触发）
PCI9018_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效（上下边沿均触发）

注明：PCI9018_TRIGDIR_POSIT_NEGAT 在边沿类型下，则表示不管是上边沿还是下边沿均触发。

TrigLevelVolt 触发电平(0-10000mV)。

ClockSource AD 时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
PCI9018_CLOCKSRC_IN	0x00	使用内部时钟
PCI9018_CLOCKSRC_PCI10M	0x01	使用主卡时钟
PCI9018_CLOCKSRC_OUT	0x02	使用外部时钟

相关函数：[CreateDevice](#) [LoadParaAD](#) [SaveParaAD](#)
[ReleaseDevice](#)

第二节、AD 状态参数结构 (PCI9018_STATUS_AD)

Visual C++ & C++Builder:

```
typedef struct _PCI9018_STATUS_AD
{
    LONG bADEanble;
```

```

LONG bTrigger;
LONG bComplete;
LONG bAheadTrig;
LONG lEndAddr;
} PCI9018_STATUS_AD, *PCI9018_STATUS_AD;

```

Visual Basic :

Private Type PCI9018_STATUS_AD

bADEnable As Long

bTrigger As Long

bComplete As Long

bAheadTrig As Long

lEndAddr As Long

End Type

Delphi:

Type // 定义结构体数据类型

PCI9018_STATUS_AD = ^PCI9018_STATUS_AD; // 指针类型结构

PCI9018_STATUS_AD = record // 标记为记录型

bADEnable : LongInt;

bTrigger : LongInt;

bComplete : LongInt;

bAheadTrig : LongInt;

lEndAddr: LongInt;

End;

LabVIEW:

请参考相关演示程序。

此结构体主要用于查询AD的各种状态, [GetDevStatusAD](#)函数使用此结构体来实时取得AD状态, 以便同步各种数据采集和处理过程。

bADEnableAD 是否已经使能, =TRUE: 表示已使能, =FALSE: 表示未使能。

bTrigger AD 是否被触发, =TRUE: 表示已被触发, =FALSE: 表示未被触发。

bComplete AD 是否整个转换过程是否结束, =TRUE: 表示已结束, =FALSE: 表示未结束。

bAheadTrigAD 触发点是否提前, =TRUE: 表示触发点提前, =FALSE: 表示触发点未提前。

lEndAddr 数据完成的结束地址。

相关函数: [CreateDevice](#) [GetDevStatusAD](#) [ReleaseDevice](#)

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位, 然后依其所选量程, 按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围 (mV)
±10000mV	$Volt = (20000.00/16384) * (ADBuffer[0] \& 0xFFF) - 10000.00$	[-10000, +9995.12]
±5000mV	$Volt = (10000.00/16384) * (ADBuffer[0] \& 0xFFF) - 5000.00$	[-1000, +4997.56]
0~5000mV	$Volt = (5000.00/16384) * (ADBuffer[0] \& 0xFFF)$	[0, +4997.56]
0~2500mV	$Volt = (2500.00/16384) * (ADBuffer[0] \& 0xFFF)$	[0, +2498.78]

下面举例说明各种语言的换算过程(以±10000mV 量程为例)

Visual C++&C++Builder:

```

Lsb = ADBuffer[0]&0xFFF;
Volt = (20000.00/16384) * Lsb - 10000.00;
Visual Basic:
Lsb = ADBuffer [0] And &HFFF
Volt = (20000.00/16384) * Lsb - 10000.00
Delphi:
Lsb: = ADBuffer[0] And $FFF;
Volt: = (20000.00/16384) * Lsb - 10000.00;
LabVIEW:
请参考相关演示程序。
    
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
通道号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```

typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;           // 文件头信息长度
    LONG FileType;               // 该设备数据文件共有的成员
    LONG BusType;                // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;              // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;            // 头信息版本(D15-D8=Major,D7-D0=Minijor) = 1.0
    LONG VoltBottomRange;        // 量程下限(mV)
    LONG VoltTopRange;           // 量程上限(mV)
    LONG BatCode;                // 同批文件识别码
    LONG ChannelCount;           // 通道总数
    LONG DataWidth;              // 设备的精度(分辨率)
    LONG bXorHighBit;            // 是否高位求反(为 1 则求反)
    PCI9018_PARA_AD ADPara;      // 保存硬件参数
    PCI9018_STATUS_AD ADStatus;  // 保存硬件参数
    LONG CrystalFreq;            // 晶振频率
    LONG ChannelNum;             // 通道号
    LONG HeadEndFlag;            // 头信息结束位
} FILE_HEADER, *PFILE_HEADER;
    
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第六章 上层用户函数接口应用实例

第一节、简易程序演示说明

怎样使用 DMA 方式取得 AD 数据

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统,您先点击 Windows 系统的[开始]菜单,再按下列顺序点击,即可打开基于 VC 的 Sys 工程。

[程序] [阿尔泰测控演示系统] [PCI9018 数据采集 AD 卡] [Microsoft Visual C++] [简易代码演示]
[DMA 方式]

第二节、高级程序演示说明

高级程序演示了本设备的所有功能,您先点击 Windows 系统的[开始]菜单,再按下列顺序点击,即可打开基于 VC 的 Sys 工程(主要参考 PCI9018.h 和 ADDoc.cpp)。

[程序] | [阿尔泰测控演示系统] | [PCI9018 数据采集 AD 卡] | [Microsoft Visual C++] | [高级代码演示]

其默认存放路径为:系统盘\ART\PCI9018\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

第七章 高速大容量、连续不间断数据采集及存盘技术详解

与ISA、USB设备同理,使用子线程跟踪AD转换进度,并进行数据采集是保持数据连续不间断的最佳方案。但是与ISA总线设备不同的是,PCI设备在这里不使用动态指针去同步AD转换进度,因为ISA设备环形内存池的动态指针操作是一种软件化的同步,而PCI设备不再有软件化的同步,而完全由硬件和驱动程序自动完成。这样一来,用户要用程序方式实现连续数据采集,其软件实现就显得极为容易。每次用ReadDeviceProAD_X函数读取AD数据时,那么设备驱动程序会按照AD转换进度将AD数据一一放进用户数据缓冲区,当完成该次所指定的点数时,它便会返回,当您再次用这个函数读取数据时,它会接着上一次的位置传递数据到用户数据缓冲区。只是要求每两次ReadDeviceAD之间的时间间隔越短越好。

但是由于我们的设备是通常工作在一个单CPU多任务的环境中,由于任务之间的调度切换非常平凡,特别是当用户移动窗口、或弹出对话框等,则会使当前线程猛地花掉大量的时间去处理这些图形操作,因此如果处理不当,则将无法实现高速连续不间断采集,那么如何更好的克服这些问题呢?用子线程则是必须的(在这里我们称之为数据采集线程),但这还不够,必须要求这个线程是绝对的工作者线程,即这个线程在正常采集中不能有任何窗口等图形操作。只有这样,当用户进行任何窗口操作时,这个线程才不会被堵塞,因此可以保证其正常连续的数据采集。但是用户可能要问,不能进行任何窗口操作,那么我如何将采集的数据显示在屏幕上呢?其实很简单,再开辟一个子线程,我们称之为数据处理线程,也叫用户界面线程。最初,数据处理线程不做任何工作,而是在Win32 API函数WaitForSingleObject的作用下进入睡眠状态,此时它基本不消耗CPU时间,即可保证其他线程代码有充分的运行机会(这里当然主要指数据采集线程),当数据采集线程取得指定长度的数据到用户空间时,则再用Win32 API函数SetEvent将指定事件消息发送给数据处理线程,则数据处理线程即刻恢复运行状态,迅速对这批数据进行处理,如计算、在窗口绘制波形、存盘等操作。

可能用户还要问,既然数据处理线程是非工作者线程,那么如果用户移动窗口等操作堵塞了该线程,而数据采集线程则在不停地采集数据,那数据处理线程难道不会因此而丢失采集线程发来的某一段数据吗?如果不另加处理,这个情况肯定有发生的可能。但是,我们采用了一级缓冲队列和二级缓冲队列的设计方案,足以避免这个问题。即假设数据采集线程每一次从设备上取出8K数据,那么我们就创建一个缓冲队列,在用户程序中最简单的办法就是开辟一个二维数组如ADBuffer [SegmentCount][SegmentSize],我们将SegmentSize视为数据采集线程每次采集的数据长度,SegmentCount则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成32,则这个缓冲队列实际上就是数组ADBuffer [32][8192]的形式。那么如何使用这个缓冲队列呢?方法很简单,它跟一个普通的缓冲区如一维数组差不多,唯一不同是,两个线程首先要通过改变SegmentCount字段的值,即这个下标Index的值来填充和引用由Index下标指向某一段SegmentSize长度的数据缓冲区。需要注意的是两个线程不共用一个Index下标变量。具体情况是当数据采集线程在AD部件被ReadDeviceAD初始化之后,首次采集数据时,则将自己的ReadIndex下标置为0,即用第一个缓冲区采集AD数据。当采集完后,则向数据处理线程发送消息,且两个线程的公共变量SegmentCount加1,(注意SegmentCount变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程

使用了，但是却没被数据处理线程处理掉的缓冲区数量。) 然后再接着将ReadIndex偏移至 1，再用第二个缓冲区采集数据。再将SegmentCount加 1，直到ReadIndex等于 31 为止，然后再回到 0 位置，重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自己被堵塞而没有被处理的缓冲区个数，然后逐一进行处理，最后再从SegmentCount变量中减去在所接受到的当前事件下所处理的缓冲区个数，具体处理哪个缓冲区由CurrentIndex指向。因此，即便应用程序突然很忙，使数据处理线程没有时间处理已到来的数据，但是由于缓冲区队列的缓冲作用，可以让数据采集线程先将数据连续缓存在这个区域中，由于这个缓冲区可以设计得比较大，因此可以缓冲很大的时间，这样即便是数据处理线程由于系统的偶而繁忙而被堵塞，也很难使数据丢失。而且通过这种方案，用户还可以在数据采集线程中对SegmentCount加以判断，观察其值是否大于了 32，如果大于，则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出，如果溢出即可报警。因此具有强大的容错处理。

图 7.1 便形象的演示了缓冲队列处理的方法。可以看出，最初设备启动时，数据采集线程在往 ADBuffer[0]里面填充数据时，数据处理线程便在 WaitForSingleObject 的作用下睡眠等待有效数据。当 ADBuffer[0]被数据采集线程充满后，立即给数据处理线程 SetEvent 发送通知 hEvent，便紧接着开始填充 ADBuffer[1]，数据处理线程接到事件后，便醒来开始处理数据 ADBuffer[0]缓冲。它们就这样始终差一个节拍。如虚线箭头所示。

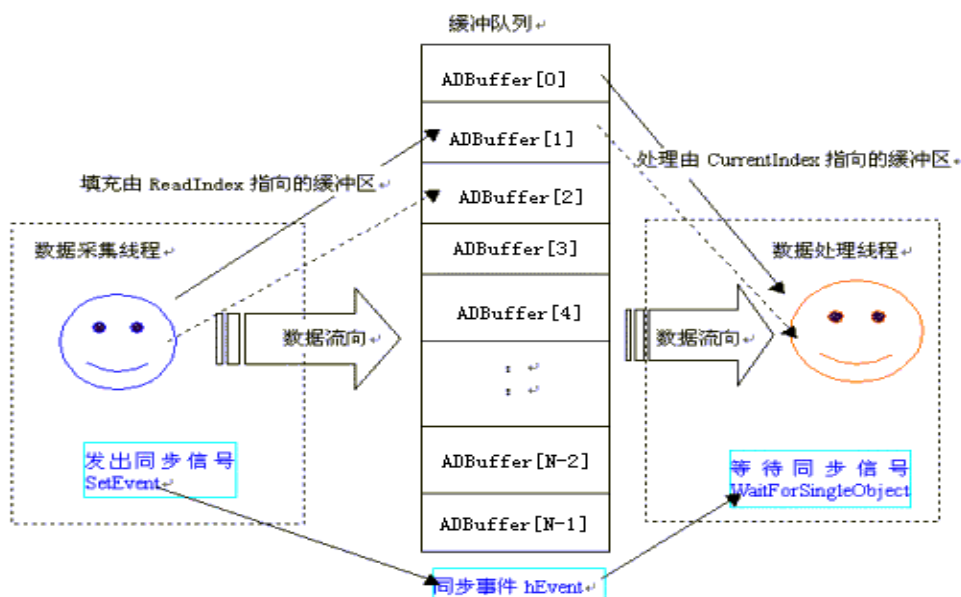


图 7.1

使用 DMA 方式实现该功能：

DMA 方式是利用直接内存存取技术实现的数据传输技术，它基本上不占用 CPU 时间就可能很快的将数据从设备读到用户缓冲区中。所以利用 DMA 方式采集数据，其吞吐率要比程序方式高很多。

需要注意的是，由于DMA方式采用了多缓冲级链的方式，因此每次接收到DMA事件后，一定要注意 [GetDevStatusAD](#) 函数返回的缓冲区状态，必须在该次事件之下，探测所有缓冲区段状态是否为新标志 1，直至所有标志为旧标志 0 后才能允许程序再去接管下一次DMA事件。

其详细应用实例及完整代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(ADDoc.h 和 ADDoc.cpp, ADThread.h 和 ADThread.cpp)。

[程序] | [阿尔泰测控演示系统] | [PCI9018 数据采集 AD 卡] | [Microsoft Visual C++] | [高级演示程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

```
void CADDoc::StartDeviceAD() // 启动线程函数
BOOL MyStartDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
UINT ReadDataThread_Dma (PVOID pThreadPara) // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara) // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
void CADDoc::StopDeviceAD() // 终止采集函数
```

第八章 共用函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数总列表（每个函数省略了前缀“PCI9018_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
GetDevVersion	获取设备固件及程序版本	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	
④ 文件对象操作函数		
CreateFileObject	初始设备文件对象	

第二节、PCI 内存映射寄存器操作函数原型说明

◆ 取得指定的指定设备寄存器组 BAR 地址

函数原型:

Visual C++ & C++ Builder:

```
BOOL GetDeviceBar (HANDLE hDevice,
                  PULONG pulPCIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceBar Lib "PCI9018" (ByVal hDevice As Long, _
                                             ByVal pulPCIBar (0 to 5) As Long) As Boolean
```

Delphi:

```
Function GetDeviceBar (hDevice : Integer;
                      pulPCIBar : Pointer) : Boolean;
StdCall; External 'PCI9018' Name 'GetDeviceBar';
```

LabVIEW:

请参考相关演示程序。

功能: 取得指定的指定设备寄存器组 BAR 地址。

参数:

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

pulPCIBar 返回 PCI BAR 所有地址。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 获取设备固件及程序版本

函数原型:

Visual C++ & C++ Builder:

```
BOOL GetDevVersion (HANDLE hDevice,
                    PULONG pulFmwVersion,
                    PULONG pulDriverVersion)
```

Visual Basic:

```
Declare Function GetDevVersion Lib "PCI9018" (ByVal hDevice As Long, _
                                             ByRef pulFmwVersion As Long, _
                                             ByRef pulDriverVersion As Long) As Boolean
```

Delphi:

```
Function GetDevVersion (hDevice : Integer;
                       pulFmwVersion: Pointer;
                       pulDriverVersion: Pointer) : Boolean;
StdCall; External 'PCI9018' Name 'GetDevVersion ';
```

LabVIEW:

请参见相关演示程序。

功能: 获取设备固件及程序版本。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pulFmwVersion 指针参数，用于取得固件版本。

pulDriverVersion 指针参数，用于取得驱动版本。

返回值: 如果执行成功，则返回 TRUE，否则会返回 FALSE。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 以单字节（即 8 位）方式写 PCI 内存映射寄存器的某个单元

函数原型:

Visual C++ & C++ Builder:

```
BOOL WriteRegisterByte( HANDLE hDevice,
                        PCHAR pbLinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

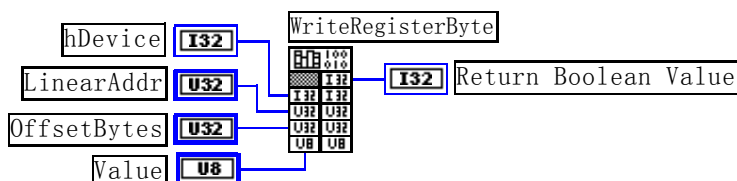
Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI9018" (ByVal hDevice As Long, _
                                                ByVal pbLinearAddr As String, _
                                                ByVal OffsetBytes As Long, _
                                                ByVal Value As Byte) As Boolean
```

Delphi:

```
Function WriteRegisterByte( hDevice : Integer;
                           pbLinearAddr: LongWord;
                           OffsetBytes : LongWord;
                           Value : Byte) : Boolean;
StdCall; External 'PCI9018' Name 'WriteRegisterByte ';
```

LabVIEW:



功能: 以单字节 (即 8 位) 方式写 PCI 内存映射寄存器。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址, 它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数, 它与 **LinearAddr** 两个参数共同确定 [WriteRegisterByte](#) 函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [WriteRegisterByte](#) [WriteRegisterWord](#)
[WriteRegisterULong](#) [ReadRegisterByte](#) [ReadRegisterWord](#)
[ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ & C++ Builder 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0))
{
    AfxMessageBox “取得设备地址失败...”;
}
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20); // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice ); // 释放设备对象

```

Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
:

```

◆ 以双字节 (即 16 位) 方式写 PCI 内存映射寄存器的某个单元

函数原型:

Visual C++ & C++ Builder:

```

BOOL WriteRegisterWord(HANDLE hDevice,
                       PCHAR pbLinearAddr,
                       ULONG OffsetBytes,
                       WORD Value)

```

Visual Basic:

```

Declare Function WriteRegisterWord Lib "PCI9018" (ByVal hDevice As Long, _
                                                ByVal pbLinearAddr As String, _
                                                ByVal OffsetBytes As Long, _
                                                ByVal Value As Integer) As Boolean

```

Delphi:

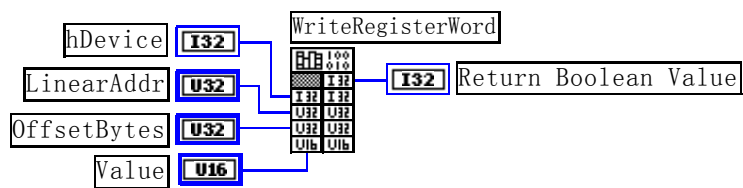
```

Function WriteRegisterWord( hDevice : Integer;
                           pbLinearAddr : LongWord;
                           OffsetBytes : LongWord;

```

Value : Word) : Boolean;
StdCall; External 'PCI9018' Name 'WriteRegisterWord';

LabVIEW:



功能: 以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数，它与 **LinearAddr** 两个参数共同确定 [WriteRegisterWord](#) 函数所访问的映射寄存器的内存单元。

Value 输出 16 位整型值。

返回值: 无。

相关函数: [CreateDevice](#) [WriteRegisterByte](#) [WriteRegisterWord](#)
[WriteRegisterULong](#) [ReadRegisterByte](#) [ReadRegisterWord](#)
[ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ & C++ Builder 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败...”;
}
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000); //往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice ); // 释放设备对象

```

Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)

```

- ◆ 以四字节（即 32 位）方式写 PCI 内存映射寄存器的某个单元
函数原型:

Visual C++ & C++ Builder:

```

BOOL WriteRegisterULong( HANDLE hDevice,
                        PCHAR pbLinearAddr,
                        ULONG OffsetBytes,
                        ULONG Value)

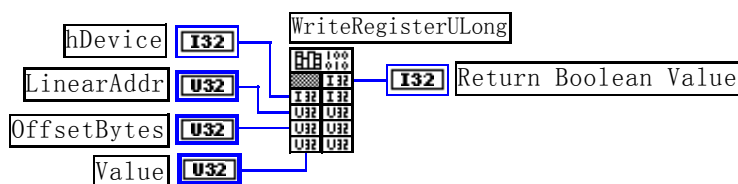
```

Visual Basic:

```
Declare Function WriteRegisterULong Lib "PCI9018" (ByVal hDevice As Long, _
                                                ByVal pbLinearAddr As String, _
                                                ByVal OffsetBytes As Long, _
                                                ByVal Value As Long) As Boolean
```

Delphi:

```
Function WriteRegisterULong(hDevice : Integer;
                           pbLinearAddr: LongWord;
                           OffsetBytes : LongWord;
                           Value : LongWord) : Boolean;
StdCall; External 'PCI9018' Name ' WriteRegisterULong ';
```

LabVIEW:

功能: 以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数，它与 **LinearAddr** 两个参数共同确定 [WriteRegisterULong](#) 函数所访问的映射寄存器的内存单元。

Value 输出 32 位整型值。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数: [CreateDevice](#) [WriteRegisterByte](#) [WriteRegisterWord](#)
[WriteRegisterULong](#) [ReadRegisterByte](#) [ReadRegisterWord](#)
[ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ & C++ Builder 程序举例:

```
:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败...”;
}
OffsetBytes=100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000); // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice ); // 释放设备对象
```

Visual Basic 程序举例:

```
:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
```

ReleaseDevice(hDevice)

:

◆ 以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元

函数原型:

Visual C++ & C++ Builder:

BYTE ReadRegisterByte(HANDLE hDevice,
PUCHAR pbLinearAddr,
ULONG OffsetBytes)

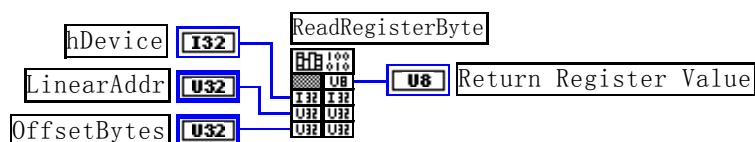
Visual Basic:

Declare Function ReadRegisterByte Lib "PCI9018" (ByVal hDevice As Long, _
ByVal pbLinearAddr As String, _
ByVal OffsetBytes As Long) As Byte

Delphi:

Function ReadRegisterByte(hDevice : Integer;
pbLinearAddr: LongWord;
OffsetBytes : LongWord) : Byte;
StdCall; External 'PCI9018' Name ' ReadRegisterByte ';

LabVIEW:



功能: 以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 [ReadRegisterByte](#) 函数所访问的映射寄存器的内存单元。

返回值: 返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数: [CreateDevice](#) [WriteRegisterByte](#) [WriteRegisterWord](#)
[WriteRegisterULong](#) [ReadRegisterByte](#) [ReadRegisterWord](#)
[ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ & C++ Builder 程序举例:

:

```
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
BYTE Value;
hDevice = CreateDevice(0); // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0); // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes); // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice ); // 释放设备对象
```

:

Visual Basic 程序举例:

:

```
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
hDevice = CreateDevice(0)
```

```

GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
:

```

◆ 以双字节（即 16 位）方式读 PCI 内存映射寄存器的某个单元

函数原型:

Visual C++ & C++ Builder:

```

WORD ReadRegisterWord( HANDLE hDevice,
                      PCHAR pbLinearAddr,
                      ULONG OffsetBytes)

```

Visual Basic:

```

Declare Function ReadRegisterWord Lib "PCI9018" (ByVal hDevice As Long, _
                                               ByVal pbLinearAddr As String, _
                                               ByVal OffsetBytes As Long) As Integer

```

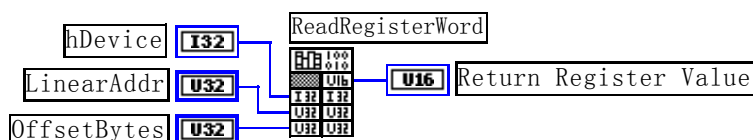
Delphi:

```

Function ReadRegisterWord(hDevice : Integer;
                          pbLinearAddr : LongWord;
                          OffsetBytes : LongWord) : Word;
StdCall; External 'PCI9018' Name 'ReadRegisterWord';

```

LabVIEW:



功能: 以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 [ReadRegisterWord](#) 函数所访问的映射寄存器的内存单元。

返回值: 返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数: [CreateDevice](#) [WriteRegisterByte](#) [WriteRegisterWord](#)
[WriteRegisterULong](#) [ReadRegisterByte](#) [ReadRegisterWord](#)
[ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ & C++ Builder 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0); // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0); // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes); // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice ); // 释放设备对象
:

```

Visual Basic 程序举例:

```

:
Dim hDevice As Long

```

```
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
:
```

◆ 以四字节（即 32 位）方式读 PCI 内存映射寄存器的某个单元

函数原型:

Visual C++ & C++ Builder:

```
ULONG ReadRegisterULONG(HANDLE hDevice,
                        PCHAR pbLinearAddr,
                        ULONG OffsetBytes)
```

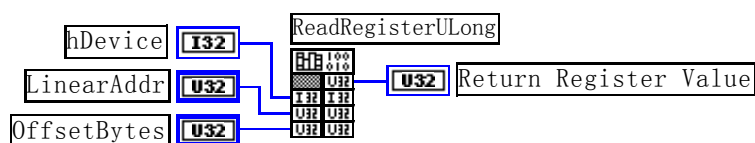
Visual Basic:

```
Declare Function ReadRegisterULONG Lib "PCI9018" (ByVal hDevice As Long, _
                                                ByVal pbLinearAddr As String, _
                                                ByVal OffsetBytes As Long) As Long
```

Delphi:

```
Function ReadRegisterULONG(hDevice : Integer;
                          pbLinearAddr : LongWord;
                          OffsetBytes : LongWord) : LongWord;
StdCall; External 'PCI9018' Name ' ReadRegisterULONG ';
```

LabVIEW:



功能: 以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对与 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 [WriteRegisterULONG](#) 函数所访问的映射寄存器的内存单元。

返回值: 返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数: [CreateDevice](#) [WriteRegisterByte](#) [WriteRegisterWord](#)
[WriteRegisterULONG](#) [ReadRegisterByte](#) [ReadRegisterWord](#)
[ReadRegisterULONG](#) [ReleaseDevice](#)

Visual C++ & C++ Builder 程序举例:

```
:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
ULONG Value;
hDevice = CreateDevice(0); // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0); // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULONG(hDevice, LinearAddr, OffsetBytes); // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice ); // 释放设备对象
:
```

Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
:

```

第三节、IO 端口读写函数原型说明

注意: 若您想在 WIN2K 系统的 User 模式中直接访问 I/O 端口, 那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动, 然后调用其中的 WritePortByteEx 或 ReadPortByteEx 等有“Ex”后缀的函数即可。

◆ 以单字节(8Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

```

BOOL WritePortByte (HANDLE hDevice,
                    PCHAR pPort,
                    BYTE Value)

```

Visual Basic:

```

Declare Function WritePortByte Lib "PCI9018" (ByVal hDevice As Long, _
                                             ByVal pPort As Long, _
                                             ByVal Value As Byte) As Boolean

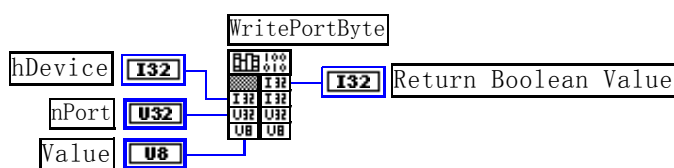
```

Delphi:

```

Function WritePortByte(hDevice : Integer;
                      pPort: LongWord;
                      Value : Byte) : Boolean;
StdCall; External 'PCI9018' Name 'WritePortByte ';

```

LabVIEW:

功能: 以单字节(8Bit)方式写 I/O 端口。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

pPort 指定寄存器的物理基地址。

Value 写入由 nPort 指定端口的值。

返回值: 若成功, 返回TRUE, 否则返回FALSE, 用户可用 [GetLastErrorEx](#) 捕获当前错误码。

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
[WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以双字(16Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

```

BOOL WritePortWord (HANDLE hDevice,
                   PCHAR pPort,
                   WORD Value)

```

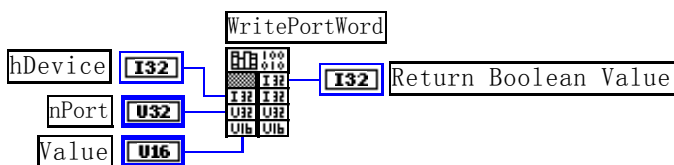
Visual Basic:

Declare Function WritePortWord Lib "PCI9018" (ByVal hDevice As Long, _
 ByVal pPort As String, _
 ByVal Value As Integer) As Boolean

Delphi:

Function WritePortWord(hDevice : Integer;
 pPort: LongWord;
 Value : Word) : Boolean;
 StdCall; External 'PCI9018' Name ' WritePortWord ';

LabVIEW:



功能: 以双字(16Bit)方式写 I/O 端口。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pPort 指定寄存器的物理基地址。

Value 写入由 nPort 指定端口的值。

返回值: 若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
 [WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以四字节(32Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

BOOL WritePortULong(HANDLE hDevice,
 PCHAR pPort,
 ULONG Value)

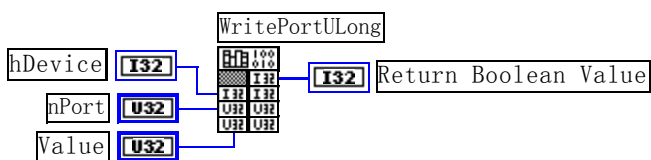
Visual Basic:

Declare Function WritePortULong Lib "PCI9018" (ByVal hDevice As Long, _
 ByVal pPort As String, _
 ByVal Value As Long) As Boolean

Delphi:

Function WritePortULong(hDevice : Integer;
 pPort: LongWord;
 Value : LongWord) : Boolean;
 StdCall; External 'PCI9018' Name ' WritePortULong ';

LabVIEW:



功能: 以四字节(32Bit)方式写 I/O 端口。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pPort 指定寄存器的物理基地址。

Value 写入由 nPort 指定端口的值。

返回值: 若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
 [WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以单字节(8Bit)方式读 I/O 端口

Visual C++ & C++ Builder:

BYTE ReadPortByte(HANDLE hDevice,
 PUCHAR pPort)

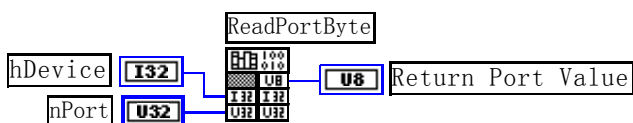
Visual Basic:

Declare Function ReadPortByte Lib "PCI9018" (ByVal hDevice As Long, _
 ByVal pPort As Long) As Byte

Delphi:

Function ReadPortByte(hDevice : Integer;
 pPort: LongWord) : Byte;
 StdCall; External 'PCI9018' Name ' ReadPortByte ';

LabVIEW:



功能: 以单字节(8Bit)方式读 I/O 端口。

参数:

hDevice设备对象句柄, 它应由[CreateDevice](#)创建。

pPort 指定寄存器的物理基地址。

返回值: 返回由 nPort 指定的端口的值。

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
 [WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以双字节(16Bit)方式读 I/O 端口

Visual C++ & C++ Builder:

WORD ReadPortWord(HANDLE hDevice,
 PUCHAR pPort)

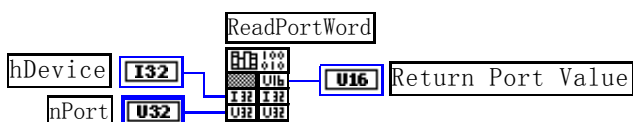
Visual Basic:

Declare Function ReadPortWord Lib "PCI9018" (ByVal hDevice As Long, _
 ByVal pPort As LongWord) As Integer

Delphi:

Function ReadPortWord(hDevice : Integer;
 pPort: LongWord) : Word;
 StdCall; External 'PCI9018' Name ' ReadPortWord ';

LabVIEW:



功能: 以双字节(16Bit)方式读 I/O 端口。

参数:

hDevice设备对象句柄, 它应由[CreateDevice](#)创建。

pPort 指定寄存器的物理基地址。

返回值: 返回由 nPort 指定的端口的值。

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
 [WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以四字节(32Bit)方式读 I/O 端口

Visual C++ & C++ Builder:

ULONG ReadPortULong(HANDLE hDevice,
PUCHAR pPort)

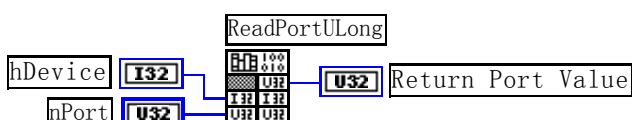
Visual Basic:

Declare Function ReadPortULong Lib "PCI9018" (ByVal hDevice As Long, _
ByVal pPort As Long) As Long

Delphi:

Function ReadPortULong(hDevice : Integer;
pPort: LongWord) : LongWord;
StdCall; External 'PCI9018' Name ' ReadPortULong ';

LabVIEW:



功能: 以四字节(32Bit)方式读 I/O 端口。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pPort 指定寄存器的物理基地址。

返回值: 返回由 nPort 指定端口的值。

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
 [WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

第四节、线程操作函数原型说明

(如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0)

◆ 创建内核系统事件

函数原型:

Visual C++ & C++ Builder:

HANDLE CreateSystemEvent(void)

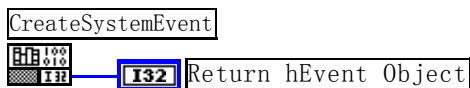
Visual Basic:

Declare Function CreateSystemEvent Lib " PCI9018 " () As Long

Delphi:

Function CreateSystemEvent() : Integer;
StdCall; External 'PCI9018' Name ' CreateSystemEvent ';

LabVIEW:



功能: 创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数: 无任何参数。

返回值: 若成功，返回系统内核事件对象句柄，否则返回 -1(或 INVALID_HANDLE_VALUE)。

◆ 释放内核系统事件

函数原型:

Visual C++ & C++ Builder:

BOOL ReleaseSystemEvent(HANDLE hEvent)

Visual Basic:

Declare Function ReleaseSystemEvent Lib " PCI9018 " (ByVal hEvent As Long) As Boolean

Delphi:

Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
StdCall; External 'PCI9018' Name ' ReleaseSystemEvent ';

LabVIEW:

请参见相关演示程序。

功能: 释放系统内核事件对象。

参数: hEvent 被释放的内核事件对象。它应由[CreateSystemEvent](#)成功创建的对象。

返回值: 若成功, 则返回 TRUE。

第五节、文件对象操作函数原型说明

◆ 创建文件对象

函数原型:

Visual C++ & C++ Builder:

HANDLE CreateFileObject (HANDLE hDevice,
LPCTSTR NewFileName,
int Mode)

Visual Basic:

Declare Function CreateFileObject Lib "PCI9018" (ByVal hDevice As Long, _
ByVal NewFileName As String, _
ByVal Mode As Integer) As Long

Delphi:

Function CreateFileObject (hDevice : Integer;
NewFileName : string;
Mode : Integer) : Integer;
Stdcall; external 'PCI9018' Name ' CreateFileObject ';

LabVIEW:

请参见相关演示程序。

功能: 初始化设备文件对象, 以期待 WriteFile 请求准备文件对象进行文件操作。

参数:

hDevice设备对象句柄, 它应由[CreateDevice](#)创建。

NewFileName 与新文件对象关联的磁盘文件名, 可以包括盘符和路径等信息。在 C 语言中, 其语法格式如: "C:\\PCI9018\\Data.Dat", 在 Basic 中, 其语法格式如: "C:\\PCI9018\\Data.Dat"。

Mode 文件操作方式, 所用的文件操作方式控制字定义如下(可通过或指令实现多种方式并操作):

常量名	常量值	功能定义
PCI9018_modeRead	0x0000	只读文件方式
PCI9018_modeWrite	0x0001	只写文件方式
PCI9018_modeReadWrite	0x0002	既读又写文件方式
PCI9018_modeCreate	0x1000	如果文件不存在可以创建该文件, 如果存在, 则重建此文件, 且清 0
PCI9018_typeText	0x4000	以文本方式操作文件

返回值: 若成功, 则返回文件对象句柄。

相关函数: [CreateDevice](#) [CreateFileObject](#)